An Improvement of the Cipolla-Lehmer Type Algorithms

Namhun Koo[1], Gook Hwa Cho[2], Byeonghwan Go[2], and Soonhak Kwon[2]
Email: nhkoo@nims.re.kr, achimheasal@nate.com, kobhh@skku.edu, shkwon@skku.edu
National Institute for Mathematical Sciences, Daejeon, Republic of Korea[1]
Sungkyunkwan University, Suwon, Republic of Korea[2]

Abstract

Let $\mathbb{F}_q$ be a finite field with q elements with prime power q and let $r > 1$ be an integer with $q \equiv 1 \pmod{r}$. In this paper, we present a refinement of the Cipolla-Lehmer type algorithm given by H. C. Williams, and subsequently improved by K. S. Williams and K. Hardy. For a given r-th power residue $c \in \mathbb{F}_q$ where r is an odd prime, the algorithm of H. C. Williams determines a solution of $X^r = c$ in $O(r^3 \log q)$ multiplications in $\mathbb{F}_q$, and the algorithm of K. S. Williams and K. Hardy finds a solution in $O(r^4 + r^2 \log q)$ multiplications in $\mathbb{F}_q$. Our refinement finds a solution in $O(r^3 + r^2 \log q)$ multiplications in $\mathbb{F}_q$. Therefore our new method is better than the previously proposed algorithms independent of the size of r, and the implementation result via SAGE shows a substantial speed-up compared with the existing algorithms.

Keywords : finite field, r-th root, Cipolla-Lehmer algorithm, Adleman-Manders-Miller algorithm, primitive root

MSC 2010 Codes : 11T06, 11Y16, 68W40

1 Introduction

Let $r > 1$ be an integer and q be a power of a prime. Finding r-th root (or finding a root of $X^r = c$) in finite field $\mathbb{F}_q$ has many applications in computational number theory and in many other related topics. Some such examples include point halving and point compression on elliptic curves [15], where square root computations are needed. Similar applications for high genus curves require r-th root computations also.

Among several available root extraction methods of the equation $X^r - c = 0$, there are two well known algorithms applicable for arbitrary integer $r > 1$; the Adleman-Manders-Miller algorithm [1], a straightforward generalization of the Tonelli-Shanks square root algorithm [16, 18] to the case of r-th root extraction, and the Cipolla-Lehmer algorithms [7, 11]. Due to the cumbersome extension field arithmetic needed for the Cipolla-Lehmer algorithm, one usually prefers the Tonelli-Shanks or the Adleman-Manders-Miller, and other related researches [2, 3, 10] exist to improve the Tonelli-Shanks.

The efficiency of the Adleman-Manders-Miller algorithm heavily depends on the exponent ν of r satisfying $r^\nu | q-1$ and $r^{\nu+1} \nmid q-1$, which becomes quite slow if $\nu \approx \log q$. Even in the case of $r = 2$, it had been observed in [14] that, for a prime $p = 9 \times 2^{3354} + 1$, running the Tonelli-Shanks algorithm using various software such as Magma, Mathematica and Maple cost roughly 5 minutes, 45 minutes, 390 minutes, respectively while the Cipolla-Lehmer costs under 1 minute in any of the above softwares. It should be mentioned that such extreme cases (of p with $p-1$ divisible by high powers of 2) may happen in some cryptographic applications.

For example, one of the NIST suggested curve [15] P-224 : $y^2 = x^3 - 3x + b$ over $\mathbb{F}_p$ uses a prime $p = 2^{224} - 2^{96} + 1$.

A generalization to r-th root extraction of the Cipolla-Lehmer square root algorithm is proposed by H. C. Williams [19] and the complexity of the proposed algorithm is $O(r^3 \log q)$ multiplications in $\mathbb{F}_q$. A refinement of the algorithm in [19] was given by K. S. Williams and K. Hardy [20] where the complexity is reduced to $O(r^4 + r^2 \log q)$ multiplications in $\mathbb{F}_q$. For the case of the square root, a new Cipolla-Lehmer type algorithm based on the Lucas sequence was given by Müller [14]. A similar result for the case $r = 3$ was also obtained by Cho et al. [5], and a possible generalization to the r-th root extraction of Müller's square root algorithm was given in [6].

In this paper, we present a new Cipolla-Lehmer type algorithm for r-th root extractions in $\mathbb{F}_q$ whose complexity is $O(r^3 + r^2 \log q)$ multiplications in $\mathbb{F}_q$, which improves previously proposed results in [19, 20]. We also compare our algorithm with those in [19, 20] using the software SAGE, and show that our algorithm performs consistently better than those in [19, 20] as is expected from the theoretical complexity estimation. In [19] and [20], only the case where r is an odd prime was considered but we will give the general arguments (i.e., no restriction on r) here.

The remainder of this paper is organized as follows: In Section 2, we briefly summarize the Cipolla-Lehmer algorithm, and introduce the works of H. C. Williams [19] and K. S. Williams and K. Hardy [20]. In Section 3, we present our refinement of the Cipolla-Lehmer algorithm. In Section 4, we give the complexity analysis of our algorithm and show the result of SAGE implementations of the three algorithms (in [19], [20], and ours). Finally, in Section 5, we give the concluding remarks.

2 Cipolla-Lehmer Algorithm in $\mathbb{F}_q$

Let q be a prime power and $\mathbb{F}_q$ be a finite field with q elements. Let $c \neq 0 \in \mathbb{F}_q$ be an r-th power residue in $\mathbb{F}_q$ for an integer $r > 1$ with $q \equiv 1 \pmod{r}$. We restrict r as an odd prime in this section.

2.1 H. C. Williams' algorithm

Let $b \in \mathbb{F}_q$ be an element such that $b^r - c$ is not an r-th power residue in $\mathbb{F}_q$. Such b can be found after r random trials of b. (See pp.479-480 in [20] for further explanation.) Then the polynomial $X^r - (b^r - c)$ is irreducible over $\mathbb{F}_q$ and there exists $\theta \in \mathbb{F}_{q^r} - \mathbb{F}_q$ such that $\theta^r = b^r - c$. Let $\omega = \theta^{q-1} = (b^r - c)^{\frac{q-1}{r}}$. Then we have $\omega^r = 1$ where ω is a primitive r-th root because $b^r - c$ is not an r-th power in $\mathbb{F}_q$.

For all $0 \leq i \leq r-1$, using $q \equiv 1 \pmod{r}$, one has $\theta^{q^i} = \theta \cdot \theta^{q^i - 1} = \theta \cdot \left(\theta^{q-1}\right)^{1+q+\cdots+q^{i-1}} = \theta\omega^i$, which implies $(b-\theta)^{q^i} = b - \theta^{q^i} = b - \omega^i\theta$. Letting $\alpha = b - \theta$, one has

$$\alpha^{\sum_{j=0}^{r-1} q^j} = (b-\theta)^{1+q+q^2+\cdots+q^{r-1}} = \prod_{i=0}^{r-1}(b - \omega^i\theta) = b^r - \theta^r = c. \tag{1}$$

Thus one may find an r-th root of c by computing $\alpha^{\frac{\sum_{j=0}^{r-1} q^j}{r}} \in \mathbb{F}_q[\theta] = \mathbb{F}_q[X]/\langle X^r - (b^r - c)\rangle$.

Proposition 1. [H. C. Williams]
Suppose that $c \neq 0$ is an r-th power in $\mathbb{F}_q$. Let $\theta^r = b^r - c$ with $\theta \in \mathbb{F}_{q^r}$ and $b \in \mathbb{F}_q$ such that

$b^r - c$ is not an r-th power in $\mathbb{F}_q$. Then letting $\alpha = b - \theta$,

$$\alpha^{\frac{\sum_{j=0}^{r-1} q^j}{r}} \in \mathbb{F}_q$$

is an r-th root of c.

The usual 'square and multiply method' (or 'double and add method' if one uses a linear recurrence relation) requires roughly $\log \frac{\sum_{j=0}^{r-1} q^j}{r} \approx r \log q$ steps for the evaluation of $\alpha^{\frac{\sum_{j=0}^{r-1} q^j}{r}}$, and therefore the complexity of the algorithm of H. C. Williams is $O(r^3 \log q)$ multiplications in $\mathbb{F}_q$. H. C. Williams' result can be expressed in Algorithm 1 using the recurrence relation technique of Section 2.2.

Algorithm 1 H. C. Williams' r-th root algorithm [19]

Input : An r-th power residue c in $\mathbb{F}_q$

Output : $x \in \mathbb{F}_q$ satisfying $x^r = c$

1: **do** Choose a random $b \in \mathbb{F}_q$ **until** $b^r - c$ is not an r-th power residue.
2: $M \leftarrow \frac{1+q+\cdots+q^{r-1}}{r}$
3: $A \leftarrow (b, -1, 0, ..., 0)$ // A is a coefficient vector of $\alpha = b - \theta$. //
4: $A \leftarrow \text{RecurrenceRelation}(A, M)$ // A is a coefficient vector of α^M. //
5: $x \leftarrow$ corresponding element of A // $x = \alpha^M$ //
6: **return** x

Note that $\alpha = b + \theta$ is used in the original paper [19], while our presentation is based on [20] where it uses $\alpha = b - \theta$. We followed [20] because it is more convenient to deal with general r which is not necessarily odd prime. For example, if one uses $\alpha = b + \theta$ as in [19], then the case of even r (such as $r = 2$) cannot be covered. Detailed explanations will be given in Section 3.

2.2 Recurrence relation

Given $\sum_{i=0}^{r-1} a_i \theta^i \in \mathbb{F}_q[\theta]$, define $a_i(j) \in \mathbb{F}_q \, (0 \le i \le r-1, \ 1 \le j)$ as

$$\sum_{i=0}^{r-1} a_i(j) \theta^i = \left(\sum_{i=0}^{r-1} a_i \theta^i \right)^j. \tag{2}$$

In particular, one has $a_i(1) = a_i$ for all $0 \le i \le r-1$. Then one has

$$\begin{aligned} \sum_{i=0}^{r-1} a_i(m+n) \theta^i &= \left(\sum_{i=0}^{r-1} a_i(m) \theta^i \right) \left(\sum_{j=0}^{r-1} a_j(n) \theta^j \right) \\ &= \sum_{l=0}^{r-1} \left(\sum_{j=0}^{l} a_j(m) a_{l-j}(n) \right) \theta^l + (b^r - c) \sum_{l=0}^{r-2} \left(\sum_{j=l+1}^{r-1} a_j(m) a_{l+r-j}(n) \right) \theta^l, \end{aligned}$$

which implies

$$a_l(m+n) = \sum_{j=0}^{l} a_j(m) a_{l-j}(n) + (b^r - c) \sum_{j=l+1}^{r-1} a_j(m) a_{l+r-j}(n) \tag{3}$$

for all $0 \le l \le r-1$. When $l = r-1$, the second summation in the equation (3) does not happen so that one has $a_{r-1}(m+n) = \sum_{j=0}^{r-1} a_j(m) a_{r-1-j}(n)$. This recurrence relation is summarized in Algorithm 2.

Algorithm 2 RecurrenceRelation(A,M)

Input : A coefficient vector $A = (a_0, a_1, \cdots, a_{r-1})$ of $a = \sum_{i=0}^{r-1} a_i \theta^i \in \mathbb{F}_q[\theta]$ and $M \in \mathbb{Z}^+$

Output : A coefficient vector of $a^M \in \mathbb{F}_q[\theta]$

Write $M = \sum M_i 2^i$ where $M_i \in \{0, 1\}$.
$(B_0, B_1, \cdots, B_{r-1}) \leftarrow (a_0, a_1, \cdots, a_{r-1})$
for k from $\lfloor \log M \rfloor - 1$ downto 0 **do**
    $(A_0, A_1, \cdots, A_{r-1}) \leftarrow (B_0, B_1, \cdots, B_{r-1})$
    for i from 0 to $r-1$ **do**
        $B_i \leftarrow \sum_{j=0}^{i} A_j A_{i-j} + (b^r - c) \sum_{j=i+1}^{r-1} A_j A_{r+i-j}$
    if $M_k = 1$ **then**
        $(A_0, A_1, \cdots, A_{r-1}) \leftarrow (B_0, B_1, \cdots, B_{r-1})$
        for i from 0 to $r-1$ **do**
            $B_i \leftarrow \sum_{j=0}^{i} A_j a_{i-j} + (b^r - c) \sum_{j=i+1}^{r-1} A_j a_{r+i-j}$
return $(B_0, \cdots, B_{r-1})$

2.3 An improvement of K. S. Williams and K. Hardy

Williams and Hardy [20] improved the algorithm of H. C. Williams by reducing the loop length to $\log q$ as follows. Write $\alpha^{\frac{\sum_{j=0}^{r-1} q^j}{r}}$ (where $\alpha = b - \theta$) as

$$\alpha^{\frac{\sum_{j=0}^{r-1} q^j}{r}} = E_1^{\frac{q-1}{r}} \cdot E_2, \tag{4}$$

where

$$E_1 = \alpha^{(q-1)^{r-2}}, \qquad E_2 = \alpha^{\frac{q^r-1}{r(q-1)} - \frac{(q-1)^{r-1}}{r}}.$$

By noticing that the exponent $\frac{q^r-1}{r(q-1)} - \frac{(q-1)^{r-1}}{r}$ of E_2 is a polynomial of q with integer coefficients and using the binomial theorem, one has the following expression of E_1 and E_2 as

$$E_1 = \prod_{i=0}^{r-2} X_i \quad \text{with } X_i = (b - \omega^i \theta)^{(-1)^{r-i}\binom{r-2}{i}}, \tag{5}$$

$$E_2 = \prod_{i=1}^{r-1} Y_i \quad \text{with } Y_i = (b - \omega^{r-i-1}\theta)^{\frac{1-(-1)^i\binom{r-1}{i}}{r}}. \tag{6}$$

Thus we have the following result of Williams and Hardy.

Proposition 2. [Williams-Hardy]

(1) Under same assumption as in Proposition 1, $E_1^{\frac{q-1}{r}} \cdot E_2$ is an r-th root of c, where

$$E_1 = \alpha^{(q-1)^{r-2}}, \qquad E_2 = \alpha^{\frac{q^r-1}{r(q-1)} - \frac{(q-1)^{r-1}}{r}}.$$

(2) E_1 and E_2 can be efficiently computed using the relations

$$E_1 = \prod_{i=0}^{r-2} (b - \omega^i \theta)^{(-1)^{r-i}\binom{r-2}{i}}, \qquad E_2 = \prod_{i=1}^{r-1} (b - \omega^{r-i-1}\theta)^{\frac{1-(-1)^i\binom{r-1}{i}}{r}}.$$

Algorithm 3 Williams-Hardy r-th root algorithm [20]

Input : An r-th power residue c in $\mathbb{F}_q$

Output : $x \in \mathbb{F}_q$ satisfying $x^r = c$

1: **do** Choose a random $b \in \mathbb{F}_q$ **until** $b^r - c$ is not an r-th power residue.
2: $\omega \leftarrow (b^r - c)^{\frac{q-1}{r}}$, where $\theta^r = b^r - c$.
3: $E_1 \leftarrow 1,\ E_2 \leftarrow 1$
4: **for** i from 1 to $r-1$ **do**
5: $\quad X_i \leftarrow (b - \omega^{i-1}\theta)^{(-1)^{r-i+1}\binom{r-2}{i-1}},\ Y_i \leftarrow (b - \omega^{r-i-1}\theta)^{\frac{1-(-1)^i\binom{r-1}{i}}{r}}$
6: $\quad E_1 \leftarrow E_1 \cdot X_i,\ E_2 \leftarrow E_2 \cdot Y_i$
7: $A \leftarrow$ coefficient vector of E_1
8: $A \leftarrow \mathrm{RecurrenceRelation}(A, \frac{q-1}{r})$
9: $E_1' \leftarrow$ corresponding element of A in $\mathbb{F}_q[\theta]$
10: $x \leftarrow E_1' \cdot E_2$
11: **return** x

The complexity of computing each of X_i in the equation (5) is of $O(\log q)+O(r)+O\left(r^2 \log \binom{r-2}{i}\right)$ multiplications in $\mathbb{F}_q$. Hence all X_i can be computed in $O(r \log q + r^4)$ $\mathbb{F}_q$-multiplications. Since the $O(r)$ multiplications of all X_i $(0 \le i \le r-2)$ in $\mathbb{F}_{q^r}$ need $O(r^3)$ multiplications in $\mathbb{F}_q$, the total complexity of computing E_1 (as a polynomial of θ degree at most $r-1$) is $O(r \log q + r^4)$ $\mathbb{F}_q$-multiplications. Similarly the complexity of computing E_2 is also $O(r \log q + r^4)$ $\mathbb{F}_q$-multiplications. For a detailed explanation, see [20]. Since the exponentiation $E_1^{\frac{q-1}{r}}$ (using the recurrence relation) needs $O(r^2 \log \frac{q-1}{r}) = O(r^2 \log q)$ multiplications in $\mathbb{F}_q$ and since the multiplication of two elements $E_1^{\frac{q-1}{r}}$ and E_2 needs $O(r)$ multiplications in $\mathbb{F}_q$ (because only the constant term of the θ expansion is needed), the total cost of computing an r-th root of c using the algorithm of K. S. Williams and K. Hardy [20] is $O(r^2 \log q + r^4)$.

3 Our New r-th Root Algorithm

In this section, we give an improved version of the Cipolla-Lehmer type algorithm by generalizing the method of [20]. Our new algorithm is applicable for all $r > 1$ with $q \equiv 1 \pmod r$. Throughout this section, we assume that r is not necessarily a prime. Thus $\omega = \theta^{q-1} = (b^r - c)^{\frac{q-1}{r}}$ may not be a primitive r-th root of unity even if $b^r - c$ is not an r-th power in $\mathbb{F}_q$. Consequently a more stronger condition is needed for the primitivity of ω. That is, ω is a primitive r-th root of unity if and only if $\omega^{\frac{r}{p}} \neq 1$ for every prime $p|r$, which holds if and only if $(b^r - c)^{\frac{q-1}{p}} \neq 1$ for every prime $p|r$. From now on, we will assume that $(b^r - c)^{\frac{q-1}{p}} \neq 1$ for every prime $p|r$ and therefore ω is a primitive r-th root of unity.

Let $\alpha \in \mathbb{F}_{q^r}$. Then, by extracting r-th roots from the following simple identity

$$\alpha^r \left(1 \cdot \alpha \cdot \alpha^{1+q} \cdots \alpha^{1+q+q^2+\cdots+q^{r-2}}\right)^q = \left(1 \cdot \alpha \cdot \alpha^{1+q} \cdots \alpha^{1+q+q^2+\cdots+q^{r-2}}\right) \alpha^{1+q+\cdots+q^{r-1}},$$

one may expect that $\alpha\left(1\cdot\alpha\cdot\alpha^{1+q}\cdots\alpha^{1+q+q^2+\cdots+q^{r-2}}\right)^{\frac{q-1}{r}}$ equals $\alpha^{\frac{1+q+\cdots+q^{r-1}}{r}}$ up to r-th roots of unity. In fact, they are exactly the same element in $\mathbb{F}_q$ and can be verified as follows;

$$\alpha^{\frac{1+q+\cdots+q^{r-1}}{r}} = \alpha^{\frac{\sum_{i=0}^{r-1} q^i}{r}} = \alpha\cdot\alpha^{\frac{(\sum_{i=0}^{r-1} q^i)-r}{r}} \tag{7}$$

$$= \alpha\cdot\alpha^{\frac{\sum_{i=0}^{r-1}(q^i-1)}{r}} = \alpha\cdot\alpha^{\frac{(q-1)\sum_{i=1}^{r-1}\sum_{j=0}^{i-1} q^j}{r}} \tag{8}$$

$$= \alpha\cdot\left(\alpha^{\sum_{i=1}^{r-1}\sum_{j=0}^{i-1} q^j}\right)^{\frac{q-1}{r}} \tag{9}$$

$$= \alpha\cdot\left(1\cdot\alpha\cdot\alpha^{1+q}\cdots\alpha^{1+q+q^2+\cdots+q^{r-2}}\right)^{\frac{q-1}{r}}. \tag{10}$$

Proposition 3. [Main Theorem]
Let $q\equiv 1 \pmod r$ with $r>1$ and let $(b^r-c)^{\frac{q-1}{p}}\neq 1$ for all prime divisors p of r. Then letting $\alpha=b-\theta$ where $\theta^r=b^r-c$,

$$\alpha\cdot\left(1\cdot\alpha\cdot\alpha^{1+q}\cdots\alpha^{1+q+q^2+\cdots+q^{r-2}}\right)^{\frac{q-1}{r}}$$

is an r-th root of c.

Based on the above simple result, we may present a new r-th root algorithm (Algorithm 4) of complexity $O(r^2\log q+r^3)$ with given information of the prime factors of r. It should be mentioned that our proposed algorithm is general in the sense that r can be any (composite) positive integer >1 satisfying $q\equiv 1 \pmod r$, while r was assumed to be an odd prime both in [19] and [20].

Both in [19] and [20], b was chosen so that $\omega=(b^r-c)^{\frac{q-1}{r}}\neq 1$, and since r is prime, ω is automatically a primitive r-th root. This property guarantees the validity of the equation (1), namely

$$(b-\theta)(b-\omega\theta)(b-\omega^2\theta)\cdots(b-\omega^{r-1}\theta)=b^r-\theta^r=c. \tag{11}$$

However if r is composite, then $\omega=(b^r-c)^{\frac{q-1}{r}}$ is not a primitive r-th root in general. In fact, letting $s>1$ be the least positive integer satisfying $\omega^s=1$, the degree of the irreducible polynomial of θ (where $\theta^r=b^r-c$) is s because

$$\theta^{q^s-1}=(\theta^{q-1})^{q^{s-1}+q^{s-2}+\cdots+q+1}=\omega^{q^{s-1}+q^{s-2}+\cdots+q+1}=\omega^s,$$

and one has

$$(b-\theta)(b-\omega\theta)\cdots(b-\omega^{r-1}\theta)=\{(b-\theta)(b-\omega\theta)\cdots(b-\omega^{s-1}\theta)\}^{\frac{r}{s}}=(b^s-\theta^s)^{\frac{r}{s}}\neq c \tag{12}$$

if $s<r$. Therefore the methods of [19] and [20] do not work for a composite r unless one assumes the primitivity of ω.

Also, even if one assumes the primitivity of $\omega=(b^r-c)^{\frac{q-1}{r}}$, one still has some problems both in [19] and [20], which will be explained in the following remarks.

Remark 1. *In [19], $\alpha=b+\theta$ was used (instead of $b-\theta$) under the assumption of $\theta^r=c-b^r$ with $(c-b^r)^{\frac{q-1}{r}}\neq 1$. If we choose $\alpha=b+\theta$ following [19], then we get*

$$(b+\theta)(b+\omega\theta)\cdots(b+\omega^{r-1}\theta)=b^r-(-\theta)^r=b^r+(-1)^{r+1}\theta^r. \tag{13}$$

Therefore if r is odd prime (as was originally assumed in [19]), one has $b^r+\theta^r=c$ and the r-th root algorithm is essentially same to the case $\alpha=b-\theta$. However when r is even (for example, when $r=2$), the original method in [19] cannot be used because $b^r+(-1)^{r+1}\theta^r=b^r-\theta^r\neq c$.

Algorithm 4 Our new r-th root algorithm

Input : An r-th power residue c in $\mathbb{F}_q$

Output : $x \in \mathbb{F}_q$ satisfying $x^r = c$

1: **do** Choose a random $b \in \mathbb{F}_q$ **until** $(b^r - c)^{\frac{q-1}{r}}$ is a primitive r-th root of unity.
2: $\omega \leftarrow (b^r - c)^{\frac{q-1}{r}}$, $\alpha \leftarrow b - \theta$ where $\theta^r = b^r - c$.
3: $P \leftarrow \alpha, A \leftarrow \alpha, W \leftarrow 1$
4: **for** $i = 1$ **to** $r - 2$ **do** $\quad //A, P \in \mathbb{F}_q[\theta]$ and $W \in \mathbb{F}_q //$
5: $\quad W \leftarrow W\omega$, $V \leftarrow b - W\theta \quad //W = \omega^i, V = b - \omega^i\theta = \alpha^{q^i} //$
6: $\quad A \leftarrow AV$, $P \leftarrow PA \quad //A = \alpha^{1+q+\cdots+q^i}, P = \alpha \cdot \alpha^{1+q} \cdots \alpha^{1+q+\cdots+q^i} //$
7: $B \leftarrow$ coefficient vector of P
8: $B \leftarrow \mathrm{RecurrenceRelation}(B, \frac{q-1}{r})$
9: $P \leftarrow$ corresponding element of B in $\mathbb{F}_q[\theta]$
10: $x \leftarrow \alpha \cdot P \quad // \ x \in \mathbb{F}_q \ //$
11: **return** x

Remark 2. *The algorithm in [20] needs E_1 and E_2 satisfying $\alpha^{\frac{\sum_{j=0}^{r-1} q^j}{r}} = E_1^{\frac{q-1}{r}} \cdot E_2$. However for composite r, E_2 cannot be well-defined in some cases, since the exponent $\frac{1-(-1)^i\binom{r-1}{i}}{r}$ in the equation (6) is not an integer in general. That is, the property $(-1)^i\binom{r-1}{i} \equiv 1 \pmod r$ only holds when r is prime. Therefore the algorithm in [20] fails to give the answer when r is composite such as $r = 4, 6, 9, \cdots$. (i.e., when $r = 4$, one has $E_2 = \alpha^{q^2 - \frac{1}{2}q}$ so the coefficient $\frac{1}{2}$ of q in the exponent is not an integer and one cannot compute E_2.) The problem of E_2 being undefined is unavoidable even if one assumes the primitivity of ω.*

4 Complexity Analysis and Comparison

4.1 Complexity analysis

An initial step of the proposed algorithm requires one to find a primitive r-th root ω in $\mathbb{F}_q$. When r is prime, one only needs to find b satisfying $\omega = (b^r - c)^{\frac{q-1}{r}} \neq 0, 1$ and the probability that a random b satisfies the required property is $\frac{1}{r} + O(q^{-\frac{1}{4}})$ ([20] pp.480) under the assumption of $r \leq q^{\frac{1}{4}}$. When r is composite, one further needs to check whether $\omega^{\frac{r}{p}} \neq 1$ for every prime divisor p of r. Since the complexity estimation $O(r^3 \log q)$ in [19] and $O(r^2 \log q + r^4)$ in [20] still hold if one assumes that a primitive root $\omega = (b^r - c)^{\frac{q-1}{r}}$ is already given, we will also assume that a primitive root ω is given in our estimation for a fair comparison.

At each i-th step of the for-loop of our proposed algorithm, step 5 needs 1 $\mathbb{F}_q$ multiplication. In step 6, the computation AV needs 1 $\mathbb{F}_{q^r}$ multiplication which, in fact, can be executed with $2r$ $\mathbb{F}_q$ multiplications because $V = b - \omega^i\theta$ is linear in θ. The computation PA needs 1 $\mathbb{F}_{q^r}$ multiplication which can be executed with r^2 $\mathbb{F}_q$ multiplications. Therefore, at the end of the for-loop, one needs at most $(r-2)(1 + 2r + r^2) < (r+1)^3$ $\mathbb{F}_q$ multiplications (of order $O(r^3)$). Since the exponentiation $P^{\frac{q-1}{r}}$ (in steps 7-9) needs $O(r^2 \log q)$ $\mathbb{F}_q$ multiplications, the total cost of our proposed algorithm is $O(r^3 + r^2 \log q)$ multiplications in $\mathbb{F}_q$. On the other hand, the cost of Algorithm 1 [19] is $O(r^2 \log \frac{q^r - 1}{r}) = O(r^3 \log q)$, and the cost of Algorithm 3 [20] is $O(r^4 + r^2 \log q)$ where $O(r^4)$ comes from the cost of computing E_1 and E_2 in steps 4-6 of

Algorithm 3. The theoretical estimation shows that our proposed algorithm is better than Algorithm 3 as r gets larger.

Finally, when $r = 2$, the for-loop can be omitted in our algorithm so that one only needs to compute $P \cdot P^{\frac{q-1}{2}}$ which is exactly same to the original Cipolla-Lehmer algorithm.

4.2 Implementation results

Table 1 shows the implementation results using SAGE of the above mentioned two algorithms and our proposed one. The implementation was performed on Intel Core i7-4770 3.40GHz with 8GB memory.

Table 1: Running time (in seconds) for r-th root algorithms

r	3	4	43	101	211
Algorithm 1 [19]	0.467	fail	2026.962	Interr.	Interr.
Algorithm 3 [20]	0.254	fail	53.849	535.043	3956.433
Our proposed algorithm	0.253	0.355	48.359	256.601	1098.401

For convenience, we used prime fields $\mathbb{F}_p$ with size about 2000 bits. Average timings of the r-th root computations for 5 different inputs of r-th power residue $c \in \mathbb{F}_p$ are computed for the primes $r = 3, 43, 101, 211$. As one can see in the table, our proposed algorithm performs better than the algorithms in [19] and [20]. The table also shows that our algorithm gets dramatically faster than other algorithms as r gets larger. For example, when $r = 101$, our algorithm is roughly 2 times faster than Algorithm 3, and when $r = 211$, our algorithm is 4 times faster than Algorithm 3. For $r = 101, 211$, the SAGE computation were interrupted after 3 hours for Algorithm 1.

5 Conclusions

We proposed a new Cipolla-Lehmer type algorithm for r-th root extractions in $\mathbb{F}_q$. Our algorithm has the complexity of $O(r^3 + r^2 \log q)$ multiplications in $\mathbb{F}_q$, which improves the previous results of $O(r^3 \log q)$ in [19] and of $O(r^4 + r^2 \log q)$ in [20]. Our algorithm is applicable for any integer $r > 1$, whereas the previous algorithms are effective only for odd prime r. Software implementations via SAGE also show that our proposed algorithm is consistently faster than the previously proposed algorithms, and becomes much more effective as r gets larger.

References

[1] L. Adleman, K. Manders and G. Miller, *On taking roots in finite fields*, Proceeding of 18th IEEE Symposium on Foundations on Computer Science (FOCS), pp. 175-177, 1977.

[2] A. O. L. Atkin, *Probabilistic primality testing*, summary by F. Morain, Inria Research Report 1779, pp.159-163, 1992.

[3] D. Bernstein, *Faster square root in annoying finite field*, preprint, *Available from* http://cr.yp.to/papers/sqroot.pdf, 2001.

[4] Z. Cao, Q. Sha, and X. Fan, *Adleman-Manders-Miller root extraction method revisited*, preprint, *available from* http://arxiv.org/abs/1111.4877, 2011.

[5] G. H. Cho, N. Koo, E. Ha, and S. Kwon, *New cube root algorithm based on third order linear recurrence relation in finite field*, to appear in Designs, Codes and Cryptography, *available from* http://link.springer.com/article/10.1007%2Fs10623-013-9910-8.

[6] G. H. Cho, N. Koo, E. Ha, and S. Kwon, *Trace expression of r-th root over finite field*, preprint, *available from* http://eprint.iacr. org/2013/041.pdf, 2013.

[7] M. Cipolla, *Un metodo per la risoluzione della congruenza di secondo grado*, Rendiconto dell'Accademia Scienze Fisiche e Matematiche, Napoli, Ser.3,Vol. IX, pp. 154-163, 1903.

[8] J. Doliskani and E. Schost, *Taking roots over high extensions of finite fields*, Mathematics of Computation, Vol.83, pp. 435-446, 2014.

[9] G. Gong and L. Harn, *Public key cryptosystems based on cubic finite field extensions*, IEEE Transactions on Information Theory, Vol.45, pp. 2601-2605, 1999.

[10] F. Kong, Z. Cai, J. Yu, and D. Li, *Improved generalized Atkin algorithm for computing square roots in finite fields*, Information Processing Letters, Vol. 98, no. 1, pp. 1-5, 2006.

[11] D. H. Lehmer, *Computer technology applied to the theory of numbers*, Studies in Number Theory, Englewood Cliffs, NJ: Pretice-Hall, pp. 117-151, 1969.

[12] R. Lidl and H. Niederreiter, *Finite Fields*, Cambridge University Press, 1997.

[13] A. J. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian, *Applications of Finite Fields*, Springer, 1992.

[14] S. Müller, *On the computation of square roots in finite fields*, Designs, Codes and Cryptography, Vol.31, pp. 301-312, 2004.

[15] NIST, *Digital Signature Standard*, Federal Information Processing Standard 186-3, National Institute of Standards and Technology, *Available from* http://csrc.nist.gov/publications/fips/, 2000.

[16] D. Shanks, *Five number-theoretic algorithms*, Proceeding of 2nd Manitoba Conference on Numerical Mathematics, Manitoba, Canada, pp. 51-70, 1972.

[17] I. Shparlinski, *Finite Fields: Theory and Computation*, Springer, 1999.

[18] A. Tonelli, *Bemerkung über die Auflösung quadratischer Congruenzen*, Göttinger Nachrichten, pp. 344-346, 1891.

[19] H. C. Williams, *Some algorithm for solving* $x^q \equiv N \pmod{p}$, Proc. 3rd Southeastern Conf. on Combinatorics, Graph Theory, and Computing (Florida Atlantic University), pp. 451-462, 1972.

[20] K. S. Williams and K. Hardy, *A refinement of H. C. Williams' qth root algorithm*, Mathematics of Computation, Vol.61, pp. 475-483, 1993.